– Optimality –
An optimisation framework for fluidity based models.

Simon Funke, Patrick Farrell

AMCG - Imperial College London

1st July, 2011

Overview

- Introduction to PDE constraint optimisation.
- A toy example.
- Optimality - An optimisation framework for fluidity based models.

Introduction

We are interested in optimisation problems with partial differential equations as constraints.

PDE-constraint optimisation problem

Find **control** g and **state** Φ such that:

$$\min_g \mathcal{J}(\Phi, g) \quad \text{subject to} \quad \mathcal{F}(\Phi, g) = 0$$

$$a \leq g \leq b$$

$\mathcal{J}$ is the **functional**, $\mathcal{F}$ is the **PDE-constraint** and a, b are the **box constraints**.

What if you want to maximise?

Example 1: Boundary control

Cool or heat the boundary of a material such that the result fits best the observation.

$$\min_g \frac{1}{2}\int_\Omega (\Phi - \Phi^{obs})^2 d\Omega + \frac{\alpha}{2}\int_{\partial\Omega} g^2 d\partial\Omega$$

subject to

$$-\Delta\Phi = 0 \quad \text{in } \Omega,$$

$$\kappa\frac{\partial\Phi}{\partial n} = g - \Phi \quad \text{on } \partial\Omega,$$

$$a \leq g \leq b$$

Φ^{obs} is an approximation f the solution. κ is the heat coefficient and a, b are upper and lower bound for the controls, respectively.

Example 2: Distributed control

Cool or heat the interior of the domain such that the result fits best the observation.

$$\min_g \mathcal{J}(\Phi, g) := \frac{1}{2}\int_\Omega (\Phi - \Phi^{obs})^2 dx + \frac{\alpha}{2}\int_{\partial\Omega} g^2 d\partial\Omega$$

subject to

$$-\Delta\Phi = g \quad \text{in } \Omega,$$

$$\Phi = 0 \quad \text{on } \partial\Omega,$$

$$a \leq g \leq b$$

Φ^{obs} is an approximation of the solution. and a, b are upper and lower bound for the controls, respectively.

More examples

- Optimal design: Optimise design parameters in the model, e.g. wing shape of an airplane.
- Parameter estimation: inverse problem that seeks unknown parameters using some a priori knowledge of the solution.
- ...

How can we solve these problems?

First optimise, then discretise

Idea:

1. Starting point is the system of first order necessary optimality conditions.
2. Discretise and solve the resulting system.

Advantages:

- Free choice of function space for the adjoint variable.
- No trouble with discontinuous discretisation methods (like slope limiters)

How can we solve these problems?

First discretise, then optimise

1. Discretise all quantities. The result is a finite dimensional optimisation problem.
2. Solve this using an optimisation algorithm.

Advantages:

- Consistent gradient

We will take this route. So how do these "optimisation algorithms" work?

First discretise then optimise

1) Discretise

$$\min_g J(\Phi, g)$$
$$F(\Phi, g) = 0$$

Assuming that for any choice of g there is exactly one state Φ, we can define $\Phi = F_g^{-1}(g)$ this system can be simplified to:

$$\min_g J(F_g^{-1}(g), g) =: \min_g \hat{J}(g). \tag{1}$$

$\hat{J}$ is called the reduced functional.

First discretise then optimise

2) Optimise

We now have a problem of the form:

$$\min_g \hat{J}(g).$$

Most efficient optimisation algorithms rely not only on functional evaluations but also on derivative information.

Reminder: If the adjoint is available, $\hat{J}'(g)$ can be computed efficiently

If λ is the adjoint solution, then the derivative of the reduced functional is given by:

$$\hat{J}'(g) = -\lambda^T \frac{\partial F}{\partial g}$$

.

General descent methods

General descent methods

- Choose an initial control g^0.
- For k=1,2, ...:
  1. If $\hat{J}'(g^k) = 0$ STOP.
  2. Choose a descent direction s, i.e. $\left\langle \hat{J}'(g^k), s^k \right\rangle < 0$.
  3. Choose a step size $\sigma_k > 0$ such that $\hat{J}(g^k + \sigma s^k) < \hat{J}(g^k)$.
  4. Set $g^{k+1} = g^k + \sigma_k s^k$.

A line search is used to determine σ.

Steepest descent method

Take the steepest descent direction for s, i.e. $s^k = -\hat{J}'(g^k)$.
Even if we choose the "optimal" step size, the steepest descent method has poor convergence properties.

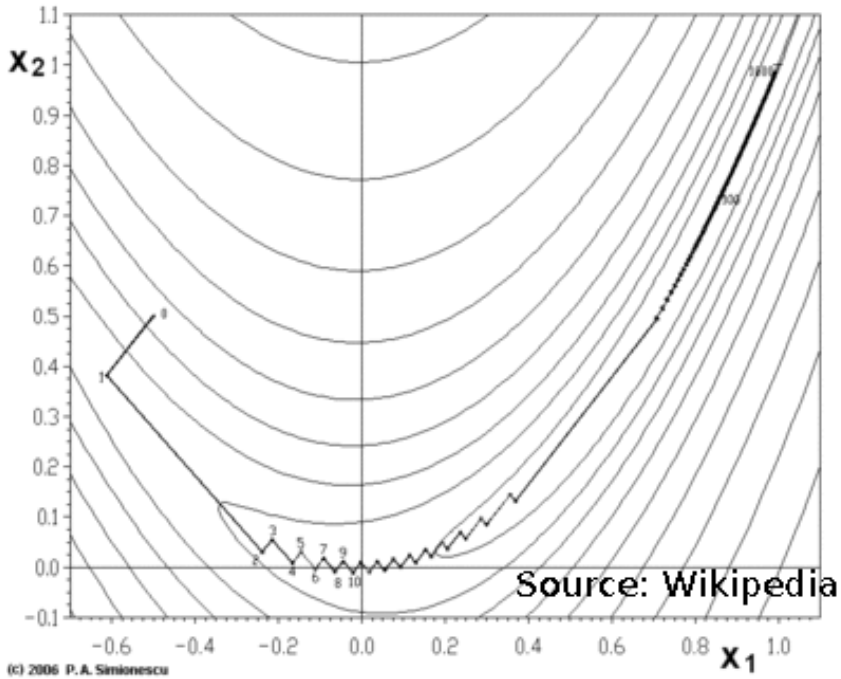


Figure: The steepest descent method applied to the Rosenbrock function.

Line search methods

Armijo rule

Find σ_k such that the Armijo conditions holds:

$$\hat{J}(g^k + \sigma s^k) - \hat{J}(g^k) \leq \sigma_k \gamma \hat{J}'(g^k) s^k.$$

Powell Wolfe rule

Find σ_k such that the Powell Wolfe conditions holds:

$$\hat{J}(g^k + \sigma s^k) - \hat{J}(g^k) \leq \sigma_k \gamma \hat{J}'(g^k) s^k,$$
$$\hat{J}'(g^k + \sigma_k s^k) s^k \geq \eta \hat{J}'(g^k) s^k.$$

with $0 < \gamma < \frac{1}{2}$ and $\gamma < \eta < 1$.

Newton method

- Define $G(g) := \hat{J}'(g)$.
- In a minimum g^* we have $\hat{J}'(g^*) = G(g^*) = 0$.
- Therefore, the Newton method for finding roots can be applied to G:

Newton method

- 0. Choose an initial control g^0.
- For k=1,2, ...:
  1. If $G(g^k) = 0$ STOP
  2. Solve the Newton equation: $G'(g^k)s^k = -G(g^k)$
  3. Set $g^{k+1} = g^k + s^k$

Note: $\hat{J}$ must be twice differentiable!

Quasi-Newton methods

- Often, computing the first derivatives is hard enough and second derivatives are not available!
- The idea of Quasi-Newton methods is to replace the Hessian of $\hat{J}(g)$ by an invertible, symmetric matrix $H(g)$.
- A very common Quasi-Newton method is the BFGS method ...

BFGS method

BFGS method

- Choose an initial control g^0 and an approximate Hessian matrix B^0.
- For k=1,2, ...:
  1. Obtain a direction s^k by solving: $B^k \mathbf{s}^k = -J'(g^k)$.
  2. Perform a line search to find an acceptable step size σ^k .
  3. Update $g^{k+1} = g^k + \sigma^k s^k$.
  4. $y^k = \hat{J}'(g^{k+1}) - \hat{J}'(g^k)$.
  5. $B^{k+1} = B^k + \frac{y^k (y^k)^{\mathrm{T}}}{\sigma^k (y^k)^{\mathrm{T}} s^k} - \frac{B^k s^k (B^k s^k)^{\mathrm{T}}}{(s^k)^{\mathrm{T}} B^k s^k}$.

A improved algorithm is obtained when the history to compute B^{k+1} is limited, known as $L - BFGS$.

Constrained optimisation

Often box constraints are known for the controls, i.e. the problem has the form:

$$\min_g \hat{J}(g)$$
$$a \leq g \leq b$$

Typical optimisation algorithms for problems with box constraints are:

- Projected gradient method
- BFGS-B, L-BFGS-B
- Sequential quadratic programming (SQP)

A manufactured toy problem

Optimisation with the 1D linear shallow water equation as constraint.

$$\min_g ||\eta(t=1) - \eta^{obs}||^2_{L_2}$$

subject to

$$\begin{aligned} \frac{\partial u}{\partial t} + G\frac{\partial \eta}{\partial x} &= 0 \\ \frac{\partial \eta}{\partial t} + H\frac{\partial u}{\partial x} &= 0 \end{aligned} \tag{2}$$

periodic boundary conditions

$$u(t=0) = u_0$$

$$\eta(t=0) = g$$

G and H are the gravity constant and the mean layer thickness, respectively.

A manufactured toy problem

Choose $G = H = 1$, $\Omega = [0, 1]$ and

$$u_0(x) = \sin(2\pi x)$$
$$\eta^{obs}(x) = -\sin(2\pi(x+1))$$

Then a solution of the optimisation problem (2) is:

$$u^*(x,t) = \sin(2\pi(x+t))$$
$$\eta^*(x,t) = -\sin(2\pi(x+t))$$
$$g^*(x) = -\sin(2\pi x)$$

where $*$ denotes the optimality of the variable.

A manufactured toy problem

```
J =  0.812511269306
J =  0.754099223614
J =  0.543149616887
J =  0.143434412067
J =  0.0297244129069
J =  0.0288974277285
J =  0.0257369687544
J =  0.0154548406941
J =  0.000267266103365
J =  4.15622732858e-06
J =  4.30176575149e-09
J =  3.60119642799e-12
J =  1.99313082756e-16
J =  8.71020616069e-21
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 10
         Function evaluations: 14
         Gradient evaluations: 14
Functional value J(m):  8.71020616069e-21
Control state m:  [-1.36060845 -2.12066174 -2.07069433 -1.22979207  0.08084897  1.36060845
  2.12066174  2.07069433  1.22979207 -0.08084897 -1.84711171 -0.77472629
 -2.21396324 -1.73515606 -0.59357824  0.77472629  1.84711171  2.21396324
  1.73515606  0.59357824]
Optimisation finished in  24.1607167721 seconds
```

Figure: Optimisation algorithm: BFGS, Runtime: 24.2s

A manufactured toy problem

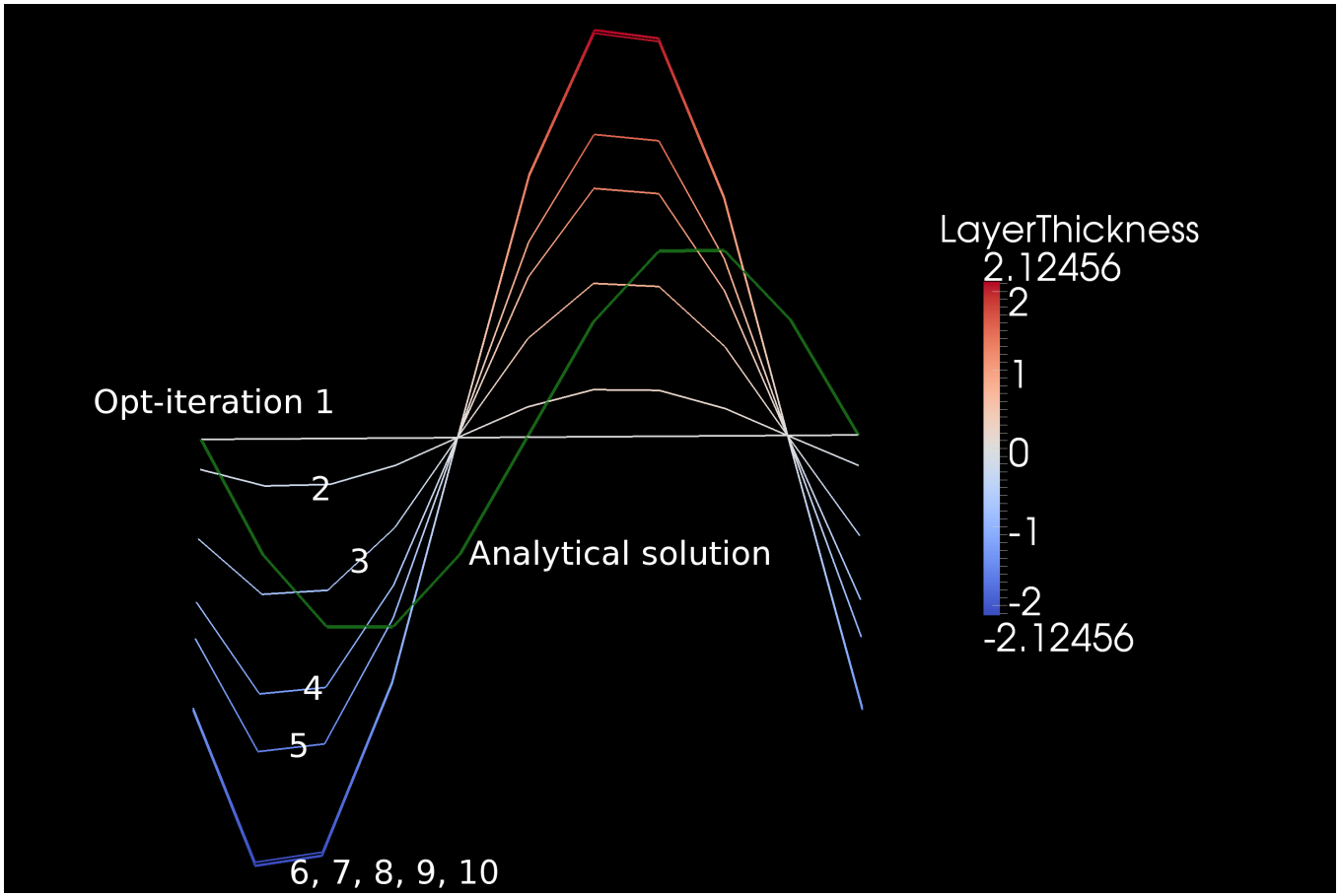


Figure: The control parameter: Initial layer thickness η.

A manufactured toy problem

Obviously we do not feed enough information into the problem. Possible solutions:

- Regularise the functional.
- Add more constraints
- Add more measurement data.

Since we know the correct answer, it is easy for us to add more measurement data. Therefore we change the functional to:

$$\min_g ||(\eta - \eta^{obs})(t = 0.5)||_{L_2}^2 + ||(\eta - \eta^{obs})(t = 1)||_{L_2}^2$$

A manufactured toy problem

```
J =  1.41152927071
J =  1.11551247267
J =  0.612349633328
J =  0.169171104779
J =  0.166080360352
J =  0.155168592946
J =  0.13545277036
J =  0.131811357361
J =  0.131811271002
J =  0.131811270557
Optimization terminated successfully.
         Current function value: 0.131811
         Iterations: 8
         Function evaluations: 10
         Gradient evaluations: 10
Functional value J(m):  0.131811270557
Control state m:  [-0.75288421 -1.37730479 -1.47564176 -1.01033372 -
  1.37730479  1.47564176  1.01033372  0.15911255 -1.129911   -0.3222
 -1.50599904 -1.30684663 -0.60852323  0.32223537  1.129911    1.5059
  1.30684663  0.60852323]
Optimisation finished in  18.4941039085 seconds
```

Figure: Optimisation algorithm: BFGS, Runtime: 18.5s

A manufactured toy problem

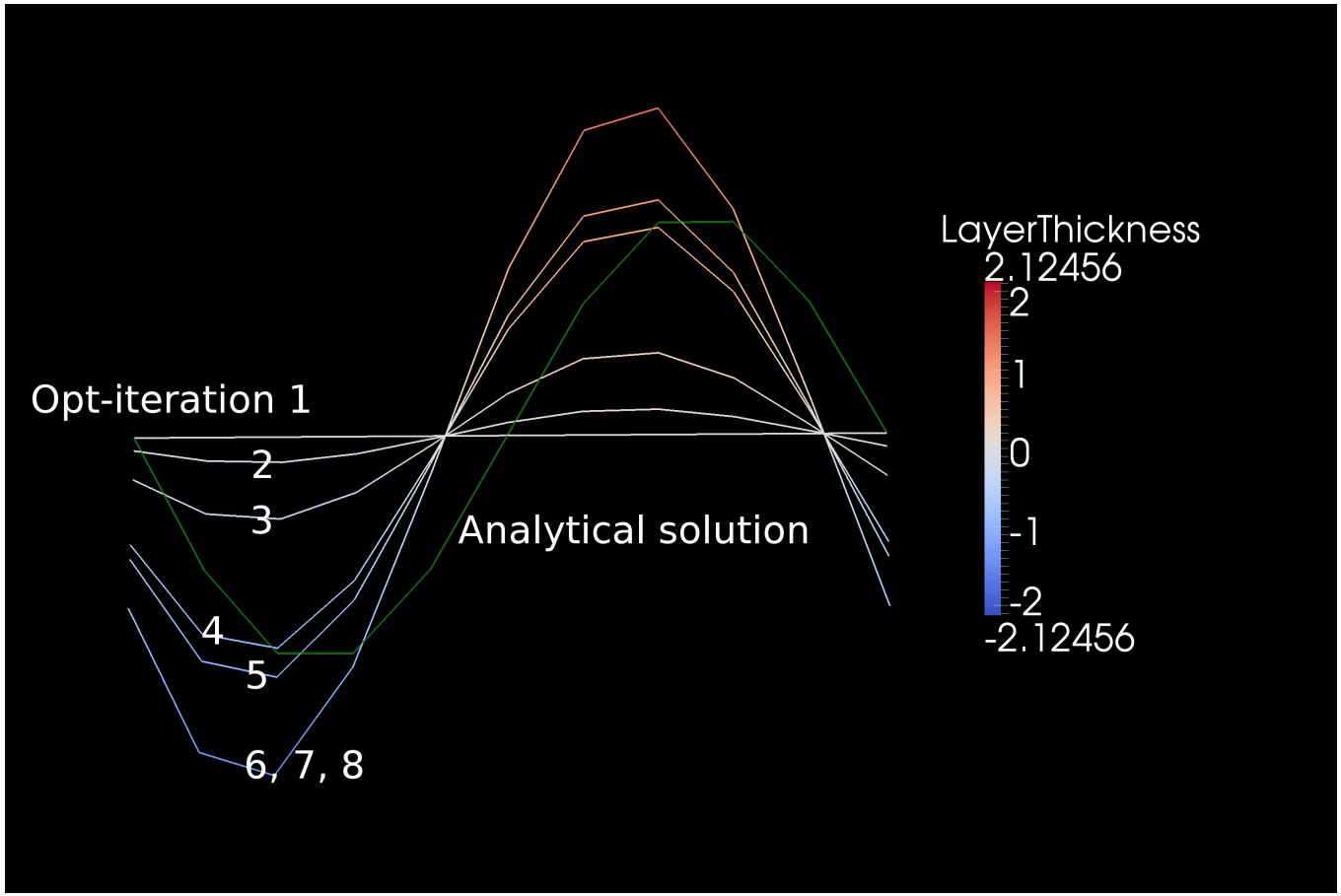


Figure: The control parameter: Initial layer thickness η.

A manufactured toy problem

- Still not perfect ...
- And suppose we do not have any other observations than at time step 1.
- But instead we know some box constraints for the control
  $=>$
  Enforce control values to be between $[-1, 1]$
  (which is true since the correct control is a sin function).

A manufactured toy problem

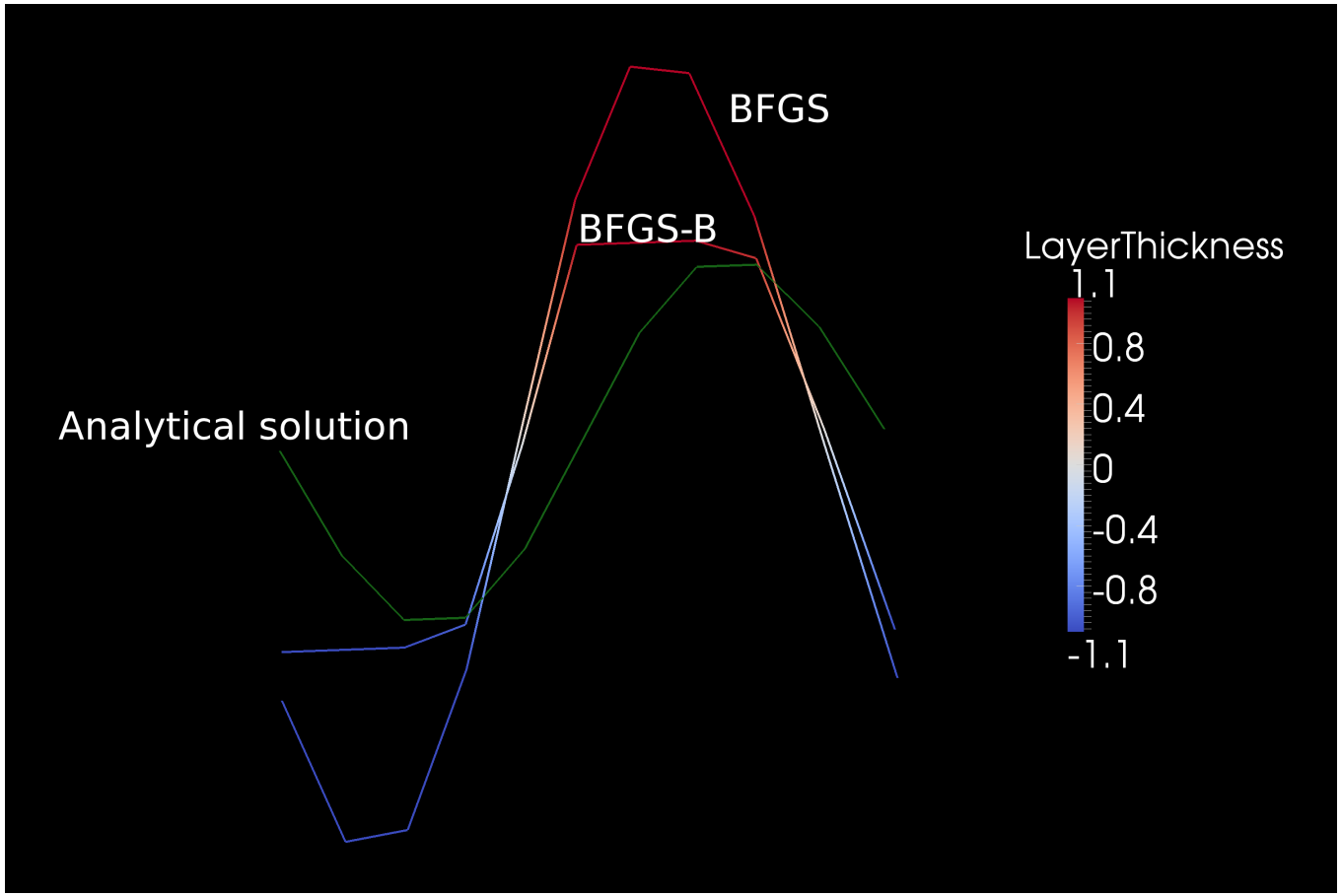


Figure: The control parameter: Initial layer thickness η.

A manufactured toy problem

Still not happy with the result? Let's add some resolution (the important one is temporal resolution for this particular problem)!

A manufactured toy problem

```
sf1409@doodson:/data/sf1409/src/fluidity_adjoint/adjoint/tests/shallow_water_optimisation$ ..
J =  1.00025139329
J =  0.945498563339
J =  0.742376339956
J =  0.316543784985
J =  0.0284537688714
J =  0.0280383010926
J =  0.0264100509351
J =  0.0204349856308
J =  0.00745378983885
J =  2.52829174042e-05
J =  2.5162107487e-05
J =  2.51613184101e-05
Optimization terminated successfully.
         Current function value: 0.000025
         Iterations: 8
         Function evaluations: 12
         Gradient evaluations: 12
Functional value J(m):  2.51613184101e-05
Control state m:  [-0.01504083 -0.09346344 -0.17130981 -0.2481     -0.32336057 -0.39662752
 -0.46744913 -0.53538875 -0.60002753 -0.66096694 -0.71783127 -0.77026993
 -0.81795962 -0.86060631 -0.89794708 -0.92975171 -0.95582411 -0.97600354
 -0.99016558 -0.99822292 -1.00012589 -0.99586274 -0.98545977 -0.96898112
 -0.94652837 -0.91823996 -0.8842903  -0.84488869 -0.80027806 -0.75073345
 -0.69656032 -0.63809267 -0.57569096 -0.50973993 -0.44064617 -0.36883569
```

Figure: Optimisation algorithm: BFGS, Runtime: 93s

A manufactured toy problem

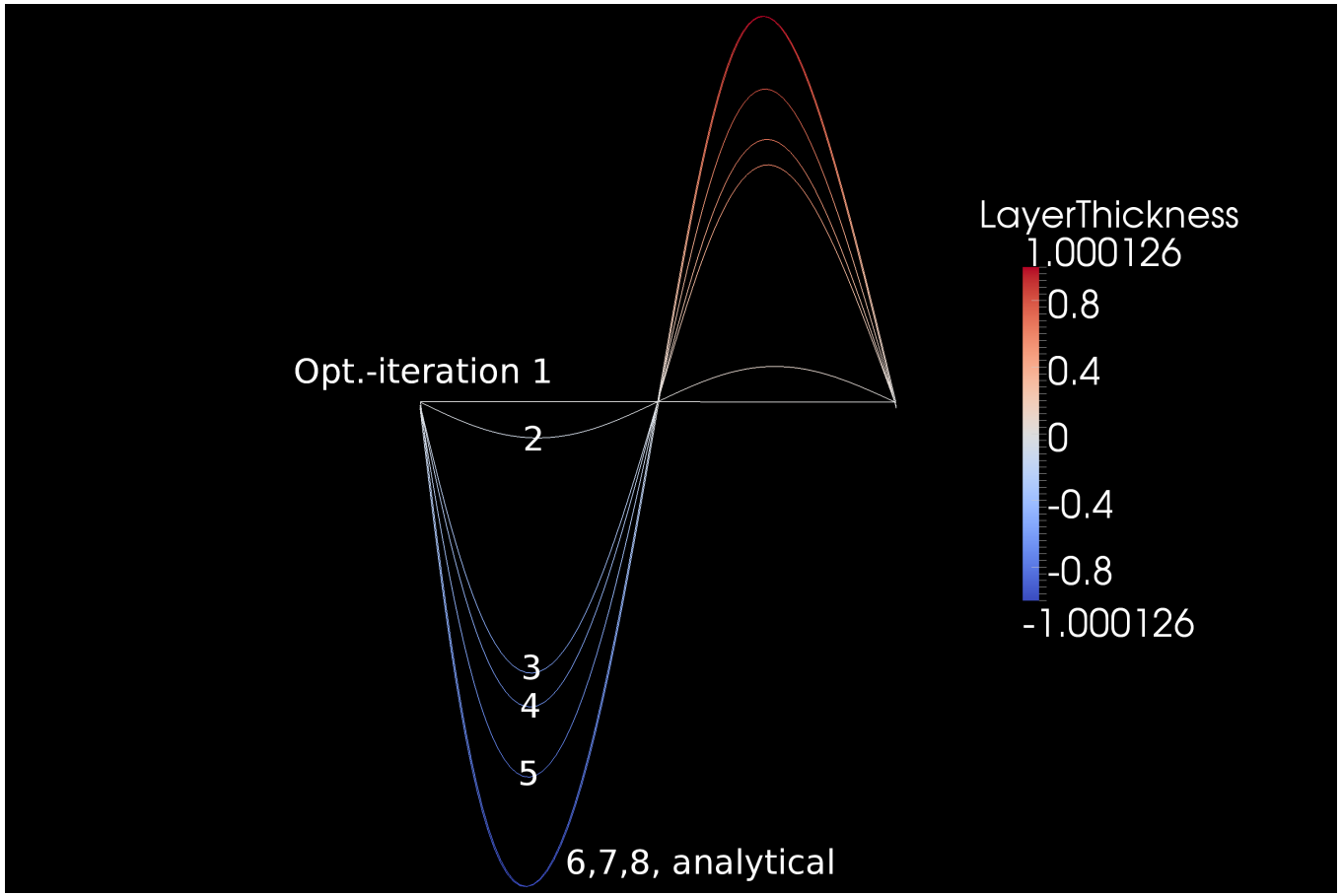


Figure: The control parameter: Initial layer thickness η.

A manufactured toy problem

Since we know the analytical solution, we can look at the convergence rate!

Convergence rate

Mesh elements	20	40	80	160
Time step	0.25	0.125	0.0625	0.03125
Rate of convergence for $\eta(t=0)$		1.87	1.93	1.97
Rate of convergence for $u(t=0)$		0	0	0 (why?)
Rate of convergence for $\eta(t=1)$		2.44	2.00	1.98
Rate of convergence for $u(t=1)$		1.87	1.93	1.98

Optimality - The optimisation framework for fluidity based models

What is it (for)?

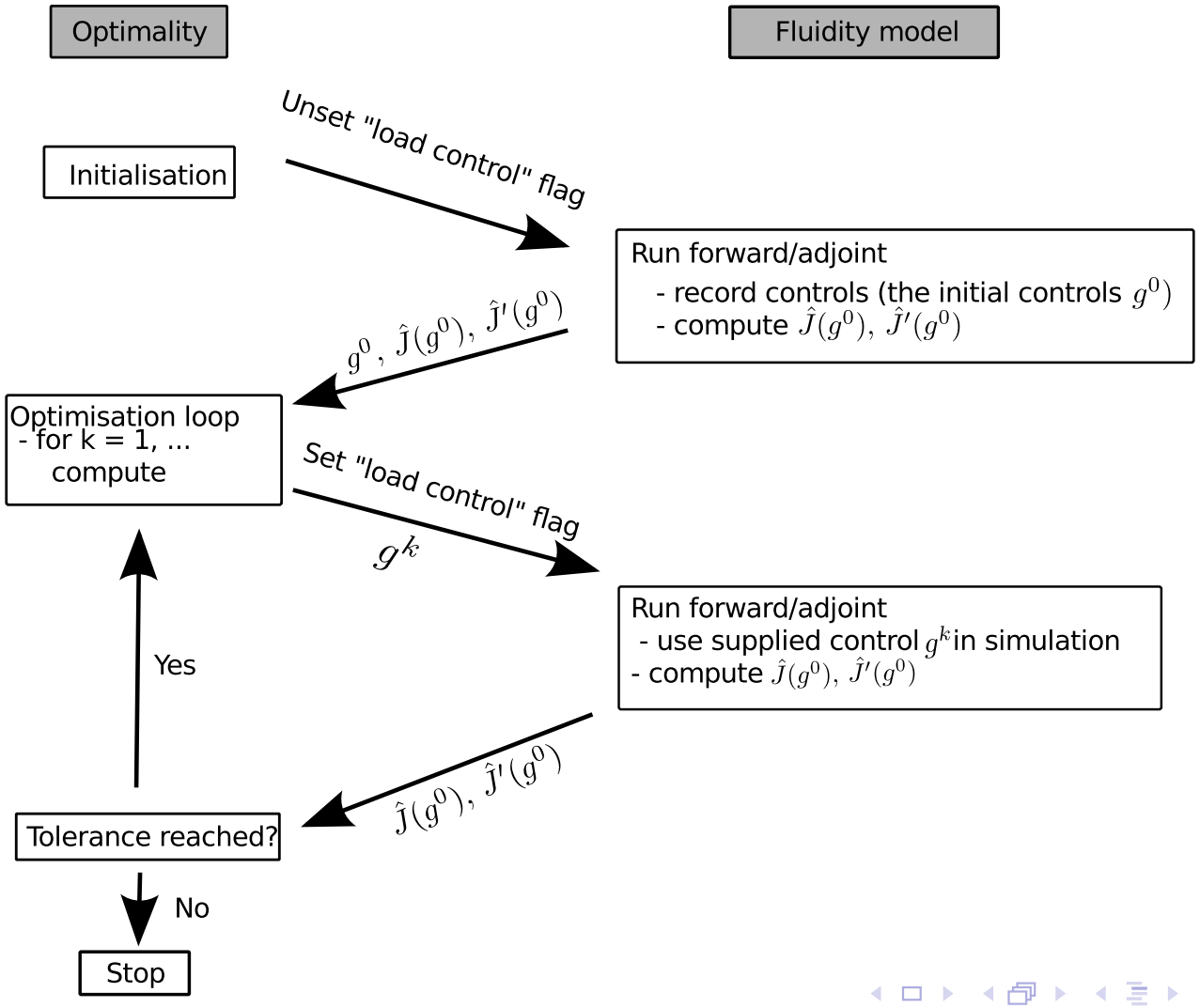
- Goal: Making the set up of optimisation problems so easy that it's fun!
- Optimality is the connector between the optimisation algorithm and the fluidity model.
- Interfaces to any fluidity based model (currently: Shallow water equation and Burgers equation).
- Interfaces to scipy.optimize which offers many optimisation algorithms (constrained and unconstrained).
- Written in python!

Optimality - The optimisation framework for fluidity based models

How does it work for the user?

1. Create a forward model diamond file:
   - Set up your forward problem as usual.
   - Code up the functional of interest (using python).
   - Select the controls you want to optimise.
2. Create an optimality diamond file:
   - Specify the location of the forward model file.
   - Choose the optimisation algorithm.
   - Select the functional/controls you want to use for the optimisation (you might have several functionals!)
3. Run optimality and watch...

Optimality - Behind the scenes



Summary and outlook

Summary and outlook:

- The shallow water and Burgers equation are "optimality-ready".
- Tests are available for the shallow water code in 1d/2d/sphere.
- Currently: An optimisation approach to accelerate the spin up time of tides on a sphere.
- An adjointed model can be made "optimality-ready" in a short time. The next steps towards an adjointed fluidity are:
  - Apply libadjoint to the nonlinear terms in the Burgers equation.
  - Start adjointing the fluidity code.

Interested? Check out lp:~fluidity-core/fluidity/adjoint